

# An improved batch processing CP approach

Sebastian Kosch

A thesis submitted in conformity with the requirements  
for the degree of *Bachelor of Applied Science*

Division of Engineering Science  
University of Toronto  
2013

# Contents

<b>1</b>	<b>Intro</b>	<b>3</b>
1.1	Chapter 8: Integer Programming . . . . .	4
1.2	Chapter 9: Building IP models I. . . . .	5
1.3	Special kinds of IP models . . . . .	6
1.4	How to formulate a good model . . . . .	7
<b>2</b>	<b>How this could be solved</b>	<b>7</b>
2.1	Possible improvements . . . . .	7

# Introduction

Books to read: Model Building in Mathematical Programming by HP Williams

Mathematical programming models all involve optimization. They want to minimize or maximize some objective function.

There are linear programming models, non-linear programming models and integer programming models.

Constraint programming and Integer Programming are twins and can usually be translated into one another. In CP each variable has a finite domain of possible values. Constraints connect/restrict the possible combinations of values which the variables can take. These constraints are of a richer variety than the linear constraints of IP and are usually expressed in the form of predicates, such as “all\_different(x1, x2, x3)”. The same condition could be imposed via IP, but it’s more troublesome to formulate. Once one of the variables has been set to one of the values in its domain (either temporarily or permanently), this predicate would imply that this value must be taken out of the domain of all other variables (“propagation”). In this way constraint propagation is used until a feasible set of values is found (or not).

CP is useful where a problem function has no objective function and we are just looking for a feasible solution. We can’t prove optimality, although we could using IP.

Comparisons and connections between IP and CLP are discussed by Barth (1995), Bockmayr and Kasper (1998), Brailsford et al. (1996), Prolland Smith (1998), Darby-Dowman and Little (1998), Hooker (1998) and Wilson and Williams (1998). The formulation of the all\_different predicate using IP is discussed by Williams and Yan (1999).

The first approach to solving a multi-objective problem is to solve the model a number of times with each objective function in turn. The result may give an idea of what to do next.

## 1.1 Chapter 8: Integer Programming

The real power of IP as a method of modelling is that of binary constraints, where variables can take on values of 0 or 1. Maybe it should be called “discrete programming.” Precise definitions of those problems that can be formulated by IP models are given by Meyer (1975) and Jeroslow (1987).

Problems with discrete inputs and outputs are the most obvious candidates for IP modelling (“lumpy inputs and outputs”). Sometimes solving an LP and rounding to the nearest integer works well, but sometimes it doesn’t, as demonstrated in Williams, first example in 8.2. The smaller the variables (say,  $<5$ ), the greater significance those rounding problems will have.

When input variables have small domains, say, machine capacities, then again this may rule out LP relaxations. A good example of an IP problem is the knapsack assignment problem. A single constraint is that the capacity of the knapsack cannot be exceeded. Any LP formulation would always fill the knapsack 100%, ignoring the discrete size of the objects. Two other well-known types of problems include *set partitioning problems* and *aircrew scheduling problems*.

Integer programming models cannot be solved directly, but need to be brute-forced in a tree search manner. The search space, however, can be greatly reduced by a number of bounding techniques often depending on the nature of the problem. That is why the general method of IP solving is called *branch-and-bound*.

So-called *cutting planes methods* usually start by solving an IP problem as LP. If the resulting solution is integral, we’re happy. Otherwise extra constraints (cutting planes) are added to the problem, further constraining it until an integer solution is found (or, if none can be found, we’re out of luck). Cutting planes make for nice illustrations they are not very efficient with large problems. Cutting planes were invented by Gomory (1958).

Enumerative methods are generally applied to pure binary problems, where the search tree is pruned. The best known of these methods is Balas’s additive algorithm described by Balas (1965). Other methods are given by Geoffrion (1969). A good overall exposition is given in Chapter 4 of Garfinkel and Nemhauser (1972).

There are so-called *pseudo-boolean methods* to solve pure binary problems

that take boolean constraints as inputs. That may be comfortable for the user sometimes, but is rarely used in practice.

Generally, branch-and-bound methods first solve the LP relaxation to check whether we're lucky enough to find an integer solution. If not, a tree search is performed.

## 1.2 Chapter 9: Building IP models I.

Binary variables are often called 01-variables. Decision variables could also have a domain like  $\{0, 1, 2\}$  or  $\{4, 12, 102\}$ . Decision variables, especially the 01 kind, can be linked to the state of continuous variables like this:  $x - M\delta \leq 0 \leftrightarrow x > 0 \rightarrow \delta = 1$ , where we know that  $x < M$  is always true.

To use a 01 variable to indicate whether the following is satisfied:

$$2x_1 + 3x_2 \leq 1 \tag{1}$$

$$x_1 \leq 1 \tag{2}$$

$$x_2 \leq 1, \tag{3}$$

so, mathematically speaking,

$$\delta = 1 \rightarrow 2x_1 + 3x_2 \leq 1 \tag{4}$$

$$2x_1 + 3x_2 \leq 1 \rightarrow \delta = 1, \tag{5}$$

Then we can argue that at most,  $2x_1 + 3x_2 = 5$ , so

$$2x_1 + 3x_2 + 4\delta \leq 5$$

will ensure that  $\delta = 1$  forces the equation to be true: use  $M = 2 + 3 - 1$  to find 4. In order to enforce  $\delta = 1$  if the equation is true, use  $m = 0 + 0 - 1$  and write

$$2x_1 + 3x_2 + \delta \geq 1$$

All kinds of logical conditions can be modelled using 01 variables, although it's not always obvious how to capture them in that format.

Logical conditions are sometimes expressed within a Constraint Logic Programming language as discussed in Section 2.4. The tightest way of expressing certain examples using linear programming constraints is described by Hooker and Yan (1999) and Williams and Yan (1999). There is a close relationship between logic and 01 integer programming, which is explained in Williams (1995), Williams and Brailsford (1999) and Chandru and Hooker (1999).

**Special Ordered Sets** Two very common types of restriction arise in mathematical programming, so two concepts (SOS1 and SOS2) have been developed. An SOS1 is a set of variables within which exactly one variable must be non-zero and the rest zero. An SOS2 is a set where at most two can be non-zero, and the two variables must be adjacent in the input ordering. Using a branch-and-bound algorithm specialized for SOS1 or SOS2 sets can speed things up greatly.

**Disjunctive constraints** It is possible to define a set of constraints and postulate that at least one of them be satisfied.

### 1.3 Special kinds of IP models

Most practical IP models do not all into any of these categories but arise as MIP models often extending an existing LP model. Here are some examples.

**Set covering problems** We have a set  $S = \{1, 2, 3, \dots, m\}$ . We have a bunch of subsets  $\mathcal{S}$  of subsets, each associated with a cost. Now cover all members of  $S$  using the least-cost members of  $\mathcal{S}$ .

**Knapsack problem** These are the really simple ones with just one constraint, namely the constraint of not being able to take more items with you than the knapsack can carry while maximizing the value of the taken objects.

**Quadratic assignment problem** Two sets of objects  $S$  and  $T$  of the same size require the objects to be matched pairwise. There are costs associated with pairs of pairs, that is the cost  $c_{ij,kl}$  is the cost of assigning  $i$  to  $j$  while also assigning  $k$  to  $l$ . This cost will be incurred if both  $\delta_{ij}$  and  $\delta_{kl}$  are true, i.e.  $\delta_{ij}\delta_{kl} = 1$ . The objective function is a quadratic expression in 01 variables:

$$\text{Minimize } \sum_{\substack{i,j,k,l=1 \\ k>l}}^n c_{ij,kl} \delta_{ij} \delta_{kl} \quad (6)$$

The quadratic version is practically unsolvable, so to be able to enumerate the possible assignments the above objective function has to be split up into separate constraints, which obviously means there will be an explosion in problem size as the number of variables grow.

## 1.4 How to formulate a good model

According to Williams, it is easy to build IP models that, while correct, are really inefficient. Fortunately, with some knowledge of what happens behind the scenes (and some practice), sucky models can often be improved. One good method to know (although it doesn't help by itself) is to turn a general integer variable into a bunch of 01 variables. Say  $\gamma$  is a general, non-negative integer variable with a known upper bound of  $u$ , then we can replace it with  $\delta_0 + 2\delta_1 + 4\delta_2 + 8\delta_3 + \dots + 2^n\delta_n$ .

**Example problem** <http://www.scribd.com/doc/49547850/Model-Building-in-Mathematical-Programming>

# How this could be solved

We're trying to create an mixed integer linear programming model (MILP). In the original paper, they had one non-linear constraint:

$$(d_{max} - d_j)(1 - x_{jk}) \geq D_k - d_j \quad \forall j \in J, \forall k \in K \quad (7)$$

This can easily be turned into

$$x_{jk}d_j - D_k \geq 0 \quad \forall j \in J, \forall k \in K \quad (8)$$

$$d_{max} - D_k \geq 0 \quad \forall k \in K \quad (9)$$

## 2.1 Possible improvements

We cannot modify the solution process itself – that's the whole point of the exercise, after all. Maybe we can preprocess to limit the domains of some decision variables. This is where “preordering” may come into play.